Department of Computer Science
University of the Western Cape

ARM Assembly Language and Programming

CSC 212 Architecture Assembly

Practical 2

2017

Venue:	
Lecturer(s):	*Prof. A Bagula*
	Dr. O Isifiade
Teaching Assistants:	*M T Allie*
	A J Henney
	A F Kruger

Practical 2 – I caught in a loop with a condition

Branching

Branching is allows for the change in the the flow of a program, which normally follows a sequential set of instructions. Branching uses the PC (Program Counter) register to effectively branch/change/jump to another instruction address in the program.

The PC register contains the address of the next instruction going to be executed. When an instruction is executed, two things may happen at the end of its execution (1) the instruction does not modify PC, it will be incremented by 4 (because in ARM, instructions are 32 bit wide, and there are 4 bytes between every instruction), (2) if the instruction modifies PC then the new value for PC is used. Table 1 displays the most common used branchin instructions.

Syntax	Semantics
B label	jump to label (unconditional)
BEQ label	jump to label if previously compared values were equal
BNE label	jump to label if previously compared values were different
BGT label	jump to label if previously compared Rn > Rm/#imm
BGE label	jump to label if previously compared Rn >= Rm/#imm
BLT label	jump to label if previously compared Rn < Rm/#imm
BLE label	jump to label if previously compared Rn <= Rm/#imm
BL label	function call (label is the function name/entry point)
BX Rd	return from function (always as BX lr)

Table 1: List of common branch instructions.

Figure 1 displays how an unconditional branch by using the instruction B and a label here in this example. Any name could be used to indicate a label.

```
.global main
main:
        MOV R1, #55     /** Move the value 55 to R1 **/
        MOV R2, #65     /** Move the value 65 to R2 **/
        B here          /** Jump to label here **/
        MOV R2, #3      /** If values are equal, this line will be  not executed **/
here:
        ADD R0, R1, R2  /** Add the values of R1 and R2 and store the answer in R0
**/
BX LR
```

Figure 1: Unconditional Branching Example

In the Figure 1 example above the following line MOV R2, #3 will never be executed, once the assembler compiler encounter the B instruction it will know that it needs to move the PC to the next instruction address which is the laber here.

Figure 2 displays the use of a conditional branching example, where the branching instruction BEQ is used together with CMP which is a conditional instruction. A conditional branch is executed when a specific condition is met.

```
.global main
main:
        MOV R1, #55       /** Move the value 55 to R1 **/
        MOV R2, #55       /** Move the value 55 to R2 **/
        CMP R1, R2        /** Compare the values in R1 and R2 **/
        BEQ add           /** If the values are equal, then jump to label called add **/
        MOV R2, #3/** If values are equal, this line will be    executed **/
add:
        ADD R0, R1, R2    /** Add the values of R1 and R2 and store the answer in R0 **/
BX LR
```

Figure 2 : Conditional Branching Example

Control Structures

Branching, covered in the previous section allows for control structures like if-then, if-then-else and loops (while and do-while).

If-Then-Else

Figure 3 below displays the sample code snippet of the if-then-else control structure as could be found in Python, and similarly in Raspberry Pi assembler languages.

Python	Assembler

```
temperature = 45;
if temperature > 70:
    print('Wear shorts.')
else:
    print('Wear long pants.')
    print('Get some exercise outside.')
```
	```
	main
	    MOV R0, #45
	    MOV R1, #70
	    CMP R0, R1
	    BGT is_equal
	not_equal :
	    mov r0, #2
	    b end
	is_equal:
	    mov r0, #1
	end:
	    bx lr
	```

Figure 3: If-Then Example

Question 1 – Its getting HOT in here

a) Write an assembler program that will request the user to input his/her name and surname and display it to the screen.

   Hint (C-External Functions : Scanf and Printf)

b) Adapt the assembler program in Figure 3 to request a temperature value (between 10 – 100), and based on the input display the correct output to the screen. Expected output :

   - Wear shorts. **Or**
   - Wear long pants and Get some exercise outside.

Loops

Python	Assembler

num = 22
if num < 0:
  print("Enter a positive number")
else:
  sum = 0
  while(num > 0):
    sum += num
    num -= 1
  print("The sum is",sum)
	.text
	.global main
	main:
	        mov r1, #0
	        mov r2, #1
	loop: cmp r2, #22
	        bgt end
	        add r1, r1, r2
	        add r2, r2, #1
	   b loop
	end:
	mov r0, r1
	bx lr

Figure 4: While Loop Example

The code snippet in Figure 4 sums the values between 1 and 22, where R2 is used as the counter and answer of the sum stored in R1. The final sum is moved to R0 and displayed.

R2 is compared to 22 (BGT R2, #22) after each loop through, and only ends (branch) when R2 is greater than 22 and jumps to the end label. Also with every loop R2 is added to R1 and stored in R1.

Question 2 – Looping around

a) Write an assembler program that calculate the Fibonacci number for a given position in a Fibonacci series e.g.

Fib(10) = 89

b) Write an assembler program to find the maximum number of the following sequence:

```
3,67,34,222,45,75,54,34,44,33,22,11,66,0
```